University of Saskatchewan
Department of Computer Science

Cmpt 330
Midterm Examination

October 29, 2001

Time: 50 minutes
Total Marks: 50

Professor: A. J. Kusalik
Closed Book

Name: _____

Student Number: _____

Directions:

Answer each of the following questions in the space provided in this exam booklet. If you must continue an answer (e.g. in the extra space on the last page, or on the back side of a page), make sure you clearly indicate that you have done so and where to find the continuation.

Make all written answers legible; no marks can be given for answers which cannot be decrypted. Where a discourse or discussion is called for, please be concise and precise.

Use of calculators is not allowed during the exam. Fortunately, you should not need a calculator for completing any of the questions.

The context for questions is the UNIX operating system, as manifest by NetBSD, unless explicitly stated otherwise. If you find it necessary to make any assumptions to answer a question, state the assumption with your answer.

Marks for each major question are given at the beginning of that question. There are a total of 50 marks (one mark per minute) .

Good luck.

For marking use only:

A. _____/19	D. _____/5	G. _____/4
B. _____/4	E. _____/7	H. _____/3
C. _____/5	F. _____/3	
		Total: _____/50

A. ***(19 marks)***

For each of the statements below, indicate whether it is **true** ("T") or **false** ("F").

___ The first version of the UNIX operating system was developed at Bell Laboratories in 1969 by Ken Thompson.

___ There are two main streams of UNIX: the BSD stream and the LINUX stream.

___ A user invokes the *whatis(1)* command with argument "`getgid`" and gets the following output:

```
csh> whatis getgid
getgid, getegid (2) - get group process identification
```

Based on the above output, *getgid* is a kernel call (rather than a command, for example).

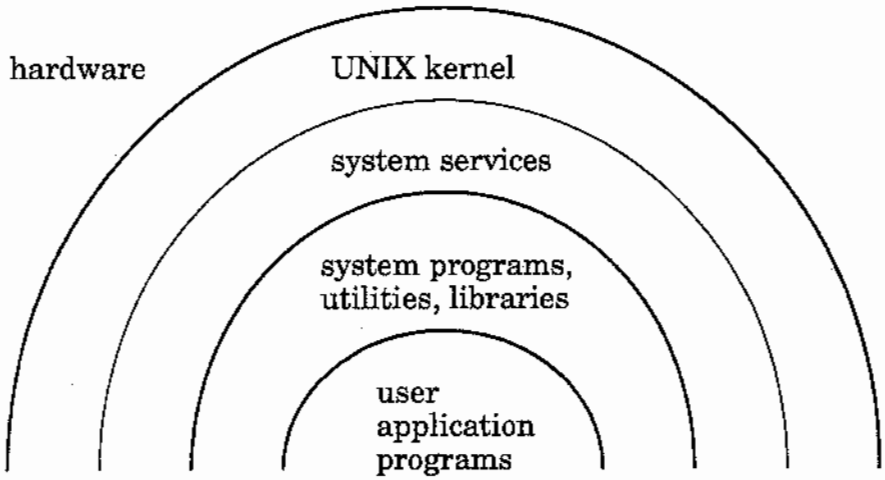
___ Consider the following C code fragment:

```
#define NAME "John"

char *s = NAME;
char *t;

t = s;
```

In the above fragment, the string `NAME` is copied to a new position in memory as a result of the assignment.

___ To put the CPU in "user mode" (as far as the mode in the hardware PSW (processor status word) register is concerned), one uses the *su* ("set user") command.

___ The following is a picture of how "layers of abstraction" are manifest in the UNIX system. Each layer defines a virtual machine to/for the layer outside it.



___ In the context of UNIX, it can be said that "a process is a program".

___ A user issues the *size* command on the file /netbsd and obtains the following:

```
tonka3> size /netbsd
text    data    bss     dec     hex     filename
4431499 393788  359784  5185071 4f1e2f  /netbsd
```

The "text" area refers to the portion of the program containing only executable instructions.

___ Consider the following program studied in class:

```
#include        <sys/types.h>
#include        <sys/wait.h>
#include        "ourhdr.h"

int main(void)
{
    char    buf[MAXLINE];
    pid_t   pid;
    int     status;

    printf("> "); /* print prompt */
    while (fgets(buf, MAXLINE, stdin) != NULL) {
        buf[strlen(buf) - 1] = 0;       /* replace newline with null */

        if ( (pid = fork()) < 0)
            err_sys("fork error");

        else if (pid == 0) {            /* child */
            execlp(buf, buf, (char *) 0);
here:       err_ret("couldn't execute: %s", buf);
            exit(127);
        }

        /* parent */
        if ( (pid = waitpid(pid, &status, 0)) < 0)
            err_sys("waitpid error");
        printf("> ");
    }
    exit(0);
}
```

Consider the *execlp(3)* call in the program above, and the statements from label here to the end of function main(). The following statement can be made: "On success of *execlp()*, the portion of the address space of the process from the point labelled by here to the end (top) of the address space, and only that portion, will be replaced by the content of the executable file named by the content of variable buf".

___ A shell variable is an environment variable whose name begins with a '$' character, as in "$path".

___ In UNIX, a signal can be thought of as a notification that an asynchronous event has occurred.

___ *logout* and *exit* are built-in commands in a (UNIX) shell. However, a user could implement a "logout" program which achieves the same functionality via the following script:

```
#!/bin/sh
kill -QUIT 0
```

___ An *sh* function which achieves the same basic functionality as the standard UNIX *mv(1)* command could be defined as follows:

```
sh> mv()
> {
> ln $1 $2
> rm $1
> }
```

___ UNIX allows for dynamic mounting and dismounting (or unmounting) of removable file systems. These file systems can come from local disks or from file servers.

___ In the context of UNIX, it can be said that "a file is an i-node".

___ The superblock (of a UNIX file system) is always one datablock in size.

___ The *dup(2)* system call is used to duplicate a process descriptor.

___ In the UNIX file system, it is possible to remove a file (from a directory) which is not open by any process, but still have the file continue to exist in the file system.

___ The following fragment of C code comes from <ufs/ffs/fs.h> (i.e. /usr/include/ufs/ffs/fs.h):

```
#define ROOTINO ((ino_t)2)
.
.
.
#define WINO ((ino_t)1)
.
.
.
typedef int32_t ufs_daddr_t;
#define NDADDR 12 /* Direct addresses in inode. */
#define NIADDR 3 /* Indirect addresses in inode. */
.
.
.
#define IFMT 0170000 /* Mask of file type. */
#define IFIFO 0010000 /* Named pipe (fifo). */
#define IFCHR 0020000 /* Character device. */
#define IFDIR 0040000 /* Directory file. */
#define IFBLK 0060000 /* Block device. */
#define IFREG 0100000 /* Regular file. */
#define IFLNK 0120000 /* Symbolic link. */
#define IFSOCK 0140000 /* UNIX domain socket. */
#define IFWHT 0160000 /* Whiteout. */
```

B. (4 marks)

In the blanks below, specify the appropriate shell redirection symbol to perform the desired function(s). The context of the question is *sh*, or *csh* with `noclobber` not set.

1. redirects the output of a command (or program) into the data input of another command (or program).

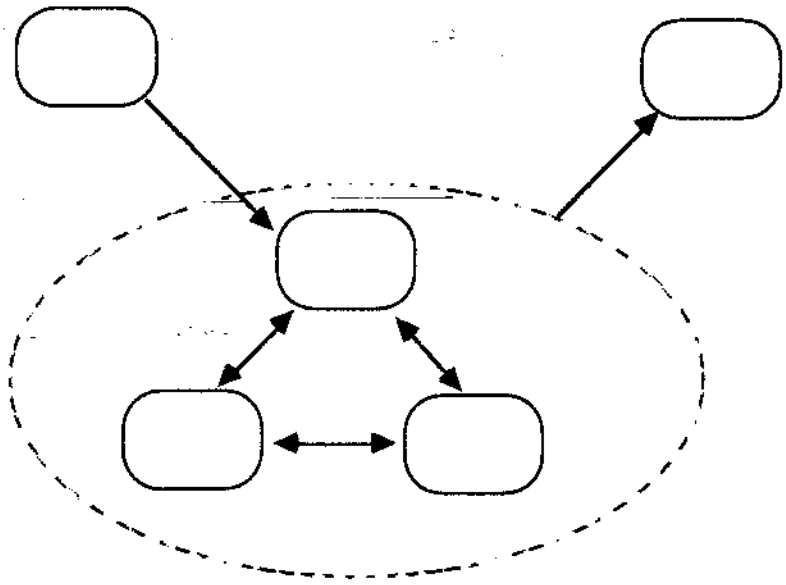
2. redirects the output of a command (or program) into a file, overwriting the file if it exists.

3. redirects the output of a command (or program) into a file, but appends the output to the file.

4. redirects the contents of a file into the standard input of a command (or program).

C. (5 marks)

Consider the states that a process might go through in its lifetime. Those states can be generalized to the following set of states: SIDL, SSLEEP, SRUN, SZOMB, SSTOP. Using these labels, fill in the following state transition diagram so that it accurately describes the states that a UNIX process may go through in its life time:



D. (5 marks)

Consider the following *sh* script:

```
#!/bin/sh
echo hi
```

Assume that the above script is in a file named `show1`. Also assume that the user's current shell is *csh*.

Now consider each of the following segments of user input (commands typed to *csh* and/or *sh*). If the input is typed to *csh*, a prompt of "`csh>`" is shown. If it is typed to *sh*, the prompt is "`sh>`". For each segment of input, indicate which of the subsequent diagrams captures the resulting process structure at the time that the *echo* command in the script is executing. Note that some of the diagrams may not be used, while some diagrams may be used more than once. There are five diagrams (a through d) to choose from.

1. `csh> show1` _____

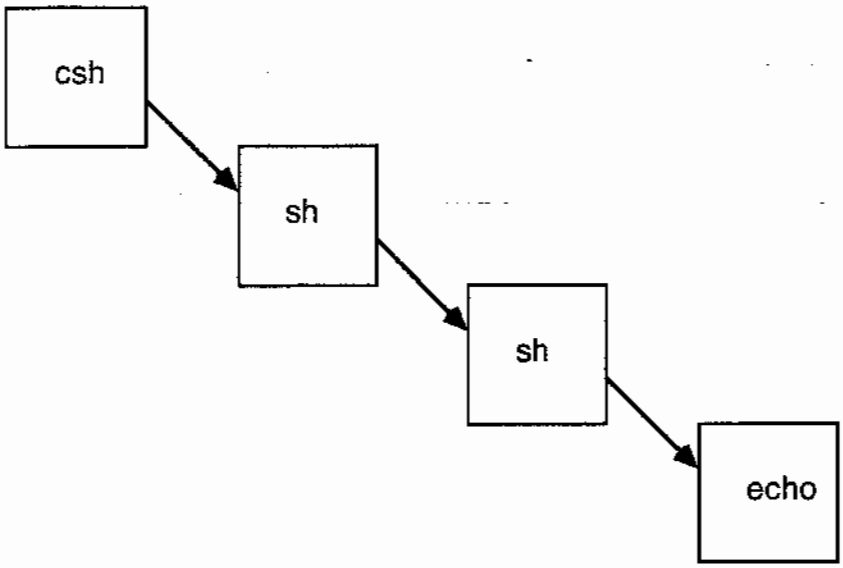
2. `csh> sh show1` _____

3. `csh> exec show1` _____

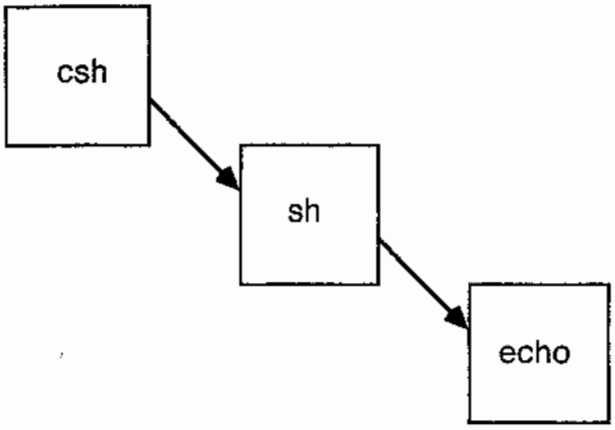
4. `csh> source show1` _____

5. `csh> sh`
`sh> show1` _____

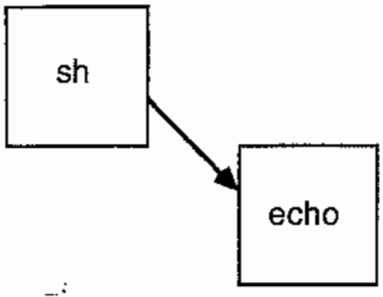
(a)



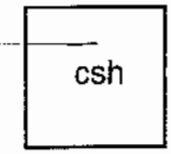
(b)



(c)



(d)



(since echo is a built-in within csh)

(e)

E. (7 marks)

Suppose your current working directory (for the current instance of your interactive shell) contains the following files:

```
log          temp.ps      test2.c
exam.ps      test1.c
```

Also assume that the shell script `show2.sh`, stored in the parent directory, has a+x permissions (anyone can execute it) and is defined as follows:

```
#!/bin/sh
echo $#
```

Finally, assume that your interactive shell is *csh*.

Consider each of the following commands given the to aforementioned interactive shell. In each case, the shell will *fork()* a new child process and *exec()* the `show2.sh` script (program) with appropriate arguments. For each command below, say what will be output by the script. I.e. in each case, what will `show2.sh` output (on its standard output)?

(a) `../show2.sh The files are: t*`

(b) `../show2.sh "The files are: *"`

(c) `../show2.sh 'The files are: t*'`

(d) `../show2.sh `echo The files are:` *`

(e) `../show2.sh -s "The files are:" *ps > log`

(f) `echo The files are: * | ../show2.sh`

(g) `../show2.sh test[123].c`

F. (3 marks)

Use this fictional (but complete) directory listing

```
tonka3# ls -al
drwxr-xr-x A dlw122  512 Jun  5 15:12 .
drwxr-xr-x B dlw122  512 Nov 11  1998 ..
-r--r--r-- 1 dlw122 5486 Jan 19  1999 dinode.h
-r--r--r-- 1 dlw122 6114 Mar  9  1999 dir.h
-r--r--r-- 1 dlw122 6569 Apr 28 15:54 inode.h
drwxr-xr-x 2 dlw122  512 Jun  5 15:12 mfs
-rwxr-xr-x 1 dlw122 2127 Jul 30 02:51 fs.h
-rwxr-xr-x 1 dlw122  322 Jul 30 02:51 dirent.h
drwxr-xr-x 2 dlw122  512 Jun  5 15:12 ufs
```

to answer the following questions:

1. What is the name for the values like **A** and **B**? I.e. what information is given by **A** and **B**, and all the values in the column immediately underneath **A** and **B**?

2. What must the value of **A** be?

3. While it would be impossible to give an exact value for **B**, we know that it is at least as large as what value? Be as specific as is possible with your answer.

G. (2+2 = 4 marks)

Answer each of the following questions with a short, precise answer.

1. Consider the return value from a *fork()* call in the child generated by the *fork()*. Can that return value be anything other than 0? If so, what other values could it be? If not, why not?

2. Cmpt330 requires that a student be able to program effectively in C. Being an effective programmer involves avoiding problems and errors, as well as being proficient in debugging.

   A student has composed the following C program:

```
#include <stdio.h>

main()
{
   long long  num;

   num = 1111222233334444;
   printf( "size of num is %d\n", sizeof( num ) );
   printf( "the number is: %d\n", num );
}
```

   Unfortunately, when the student runs the program, she gets some unexpected output:

```
tonka5> a.out
size of num is 8
the number is: -1770257748
tonka5>
```

I.e. the user expected that the value of num output would be `1111222233334444`. What is wrong with the above program? Why is it not behaving as the user expected? Note that the program does compile without errors.

H. ***(3 marks)***

A user wants to determine whether or not, on the *tonka* machines in the NetBSD, the commands *cc* and *gcc* are, in fact, the same command (i.e. that the files invoked to execute the *cc* and *gcc* commands are one in the same file). What shell command can the user give to determine unequivocally if the *cc* and *gcc* commands are the same (refer to the same file)? I.e. give a single shell command that will unambiguously determine or indicate whether the *cc* and *gcc* commands result in the same file being *exec()*-ed. Your command can be to *csh* or *sh*. However, it cannot be a script, a function, or a sequence of separate commands.

Hint: use a complex shell command involving *which(1)* and command substitution. The command can either directly indicate whether *cc* and *gcc* are the same, or provide information from which the user can directly ascertain whether they are the same. An example of the operation of *which(1)* is as follows:

```
tonka3> which cc
/usr/bin/cc
```